Differential Analysis and Fingerprinting of ZombieLoads on Block Ciphers

Till Schlüter[1*,2], Kerstin Lemke-Rust[2]

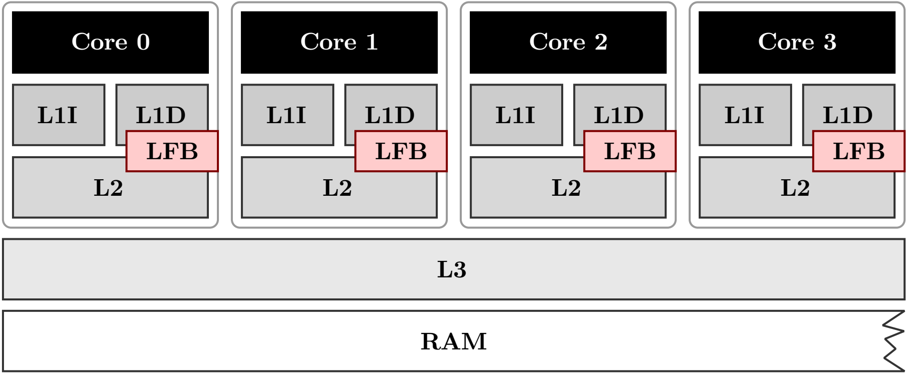
CARDIS 2020
November 19, 2020

1) CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

2) Hochschule Bonn-Rhein-Sieg University of Applied Sciences

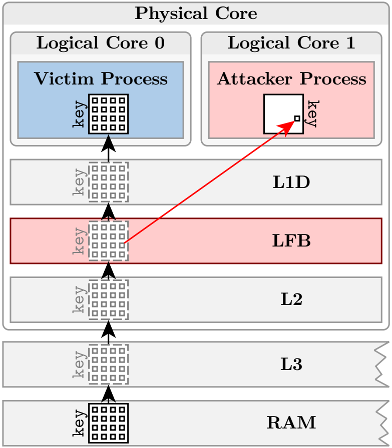
*) Also with Saarbrücken Graduate School of Computer Science.

Memory hierarchy



ZombieLoad

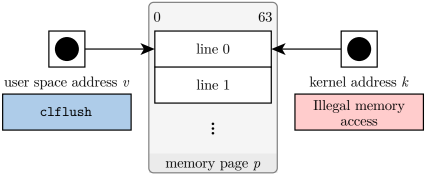
- 2 processes on SMT siblings
  - **Victim:** loads confidential data
  - **Attacker:** executes faulting load instructions
    - Transient load from LFB entry
    - Extract 1 byte at attacker-chosen index via cache-based side-channel

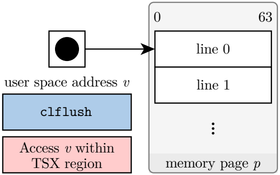


- Data sampling attack
  - Source address **unknown**
  - Time of the attack **known**

ZombieLoad: Variants

How to trigger a ZombieLoad?

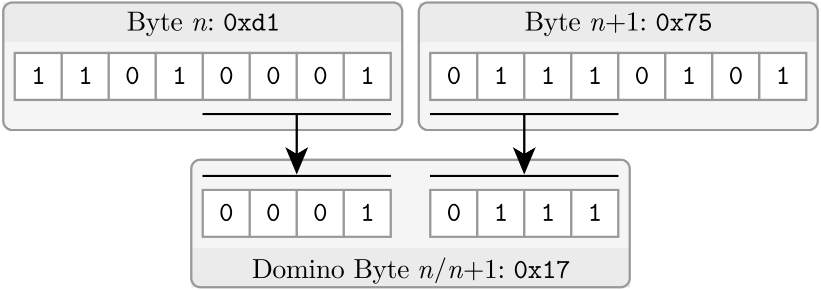




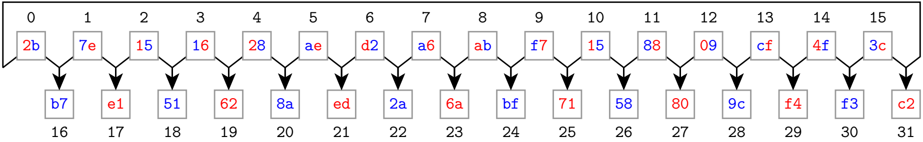
- **Variant 1:** Meltdown-like
- Requires Meltdown-vulnerable machine

- **Variant 2:** based on TSX
- Works on Meltdown-resistant machines

Leaking constant byte chains: Domino attack (1)

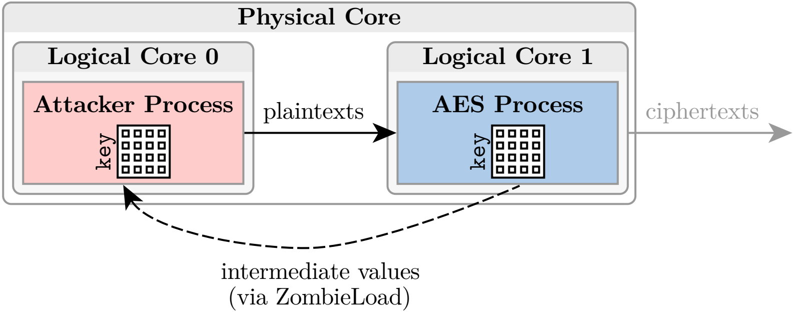


Leaking constant byte chains: Domino attack (2)



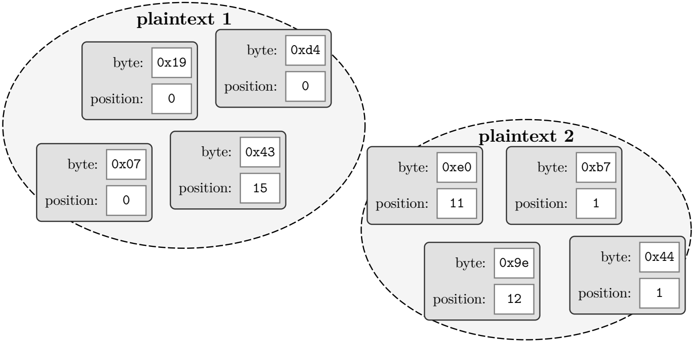
1. ZombieLoad attack (Schwarz et al., 2019)
2. Differential Attack
3. Cache Line Fingerprinting
4. Countermeasures
5. Conclusion

System/attacker model



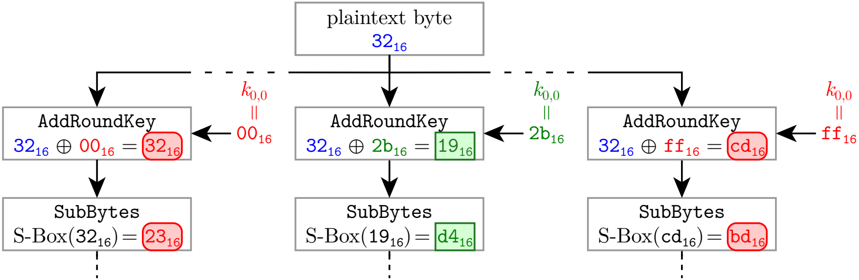
Differential attack on an AES implementation (1)

1. **Collect Samples**
   - Plaintext varies over time
   - Samples are assigned to the corresponding plaintext



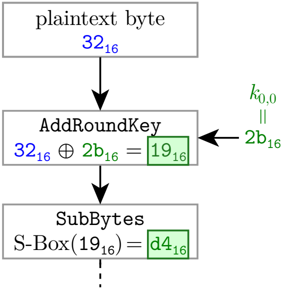
Differential attack on an AES implementation (2)

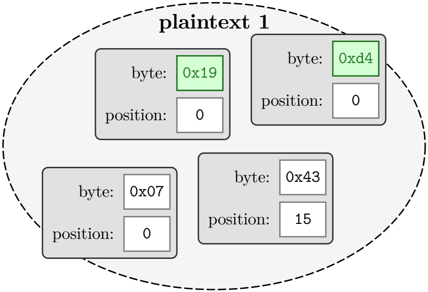
2. **Analysis**
   2.1 Calculate expected intermediate values for all plaintexts and all possible key byte hypotheses
   - byte-oriented
   - after two AES operations



Differential attack on an AES implementation (3)

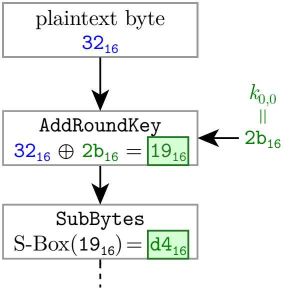
2. **Analysis**
   2.2 Find matching values

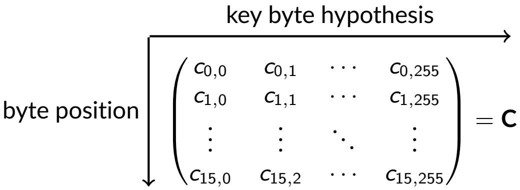




Differential attack on an AES implementation (4)

2. **Analysis**
   2.3 Increment match counter $c_{0,\mathtt{0x2b}} = c_{0,43}$ by 1.





Differential attack on an AES implementation (5)

2. **Analysis**
   2.4 Find the maximum value in each row of **C**.
   - The column index of the maximum value indicates the most probable key byte.

key byte hypothesis →

byte position ↓

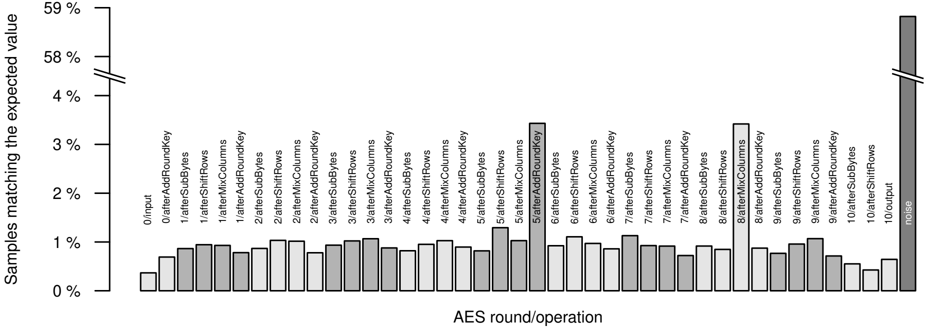
4	3	1	0	1	2	1	1	1	0	2	0	0	1	0	0	0	1	1	0	0	1	0	0	1	1	2	1	···
4	3	1	0	2	1	1	4	0	1	1	0	3	0	0	1	0	0	2	0	1	3	0	0	0	1	3	0	···
4	2	0	1	0	0	2	0	2	0	2	1	1	1	0	1	0	1	1	1	1	**16**	3	0	1	0	0	2	···
3	1	1	0	0	1	1	0	1	1	2	1	0	0	1	2	2	3	0	0	2	0	**16**	2	1	0	1	0	···
0	1	1	2	0	1	0	0	0	0	1	0	1	0	1	0	0	1	1	0	1	0	0	1	2	0	0	0	···
4	0	0	0	1	2	0	0	3	1	0	1	2	1	1	1	2	1	0	1	0	1	0	0	0	1	2	1	···
3	3	0	2	1	0	0	2	1	1	1	0	0	2	3	0	2	3	4	0	0	0	0	0	0	1	0	0	···
2	1	0	1	1	1	2	1	1	0	0	1	1	1	0	1	0	2	1	0	2	0	1	0	1	1	0	0	···
5	1	1	0	0	1	2	0	1	0	1	2	1	1	1	2	1	1	1	0	1	2	0	0	1	0	0	0	···
2	1	2	1	1	2	0	0	0	0	0	3	1	1	1	3	0	1	0	1	0	0	0	2	1	0	0	1	···
6	0	0	2	2	1	0	2	0	1	0	0	2	1	0	0	0	0	1	1	2	**16**	1	1	1	1	1	1	···
1	1	1	0	0	0	0	1	1	0	1	0	1	1	1	1	1	0	0	0	1	0	1	1	0	2	0	0	···
13	11	10	9	7	11	9	10	10	**16**	11	10	14	7	10	9	10	11	10	5	9	5	6	11	9	10	7	11	···
3	0	0	0	2	1	2	0	1	0	1	0	0	1	0	2	0	4	1	2	0	1	0	0	2	0	2	0	···
3	5	0	1	2	0	1	0	1	1	1	2	2	0	2	1	1	0	0	1	0	0	0	0	1	2	0	0	···
3	0	0	0	0	0	0	1	2	1	1	1	0	2	1	0	0	1	1	0	1	1	1	1	1	0	1	1	···

$= \mathbf{C}$

Case study

- Apply the algorithm to a byte-oriented AES implementation in C
- *aes-min*: `https://github.com/cmcqueen/aes-min`

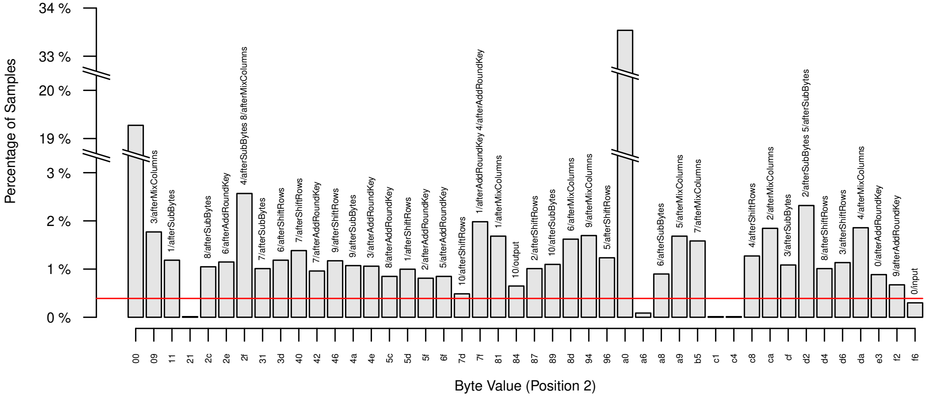
Assignment of samples to AES operations



(i7-2620M, variant 1)

- Samples contain AES intermediate results from all operations in all rounds.
- Side note: sampling rate $<<$ frequency of AES computations
  $\Rightarrow$ ≤ 1 samples per AES computation

Distribution of samples



(i7-2620M, variant 1)

- Most samples can be assigned to AES operations
- Only few additional values due to noise

Results (1)

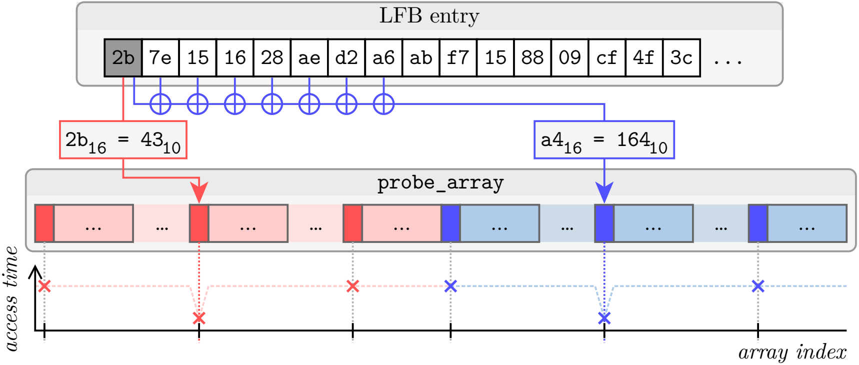
CPU	Variant	No. of samples	Samples/ plaintext	Avg. duration (s)	Avg. key bytes	Full key recoveries	
i3-2120	1	30,000	500	3.4	14.7	1/10	(10%)
		100,000	1,000	10.0	16.0	10/10	(100%)
i7-2620M	1	60,000	8,000	6.2	14.9	1/10	(10%)
		200,000	4,000	53.7	16.0	10/10	(100%)
i5-4300M	1	20,000	1,000	8.0	13.2	1/10	(10%)
		200,000	4,000	65.4	16.0	10/10	(100%)
E3-1270v6	1	3,000	500	732.7	0	0/10	(0%)
i7-8650U	1	3,000	500	1,033.4	0	0/10	(0%)
E3-1270v6	2	800,000	300	405.1	11.7	0/10	(0%)
i7-8650U	2	600,000	1,000	122.3	14.8	4/10	(40%)
		800,000	300	197.2	15.8	8/10	(80%)

Demo: Differential attack

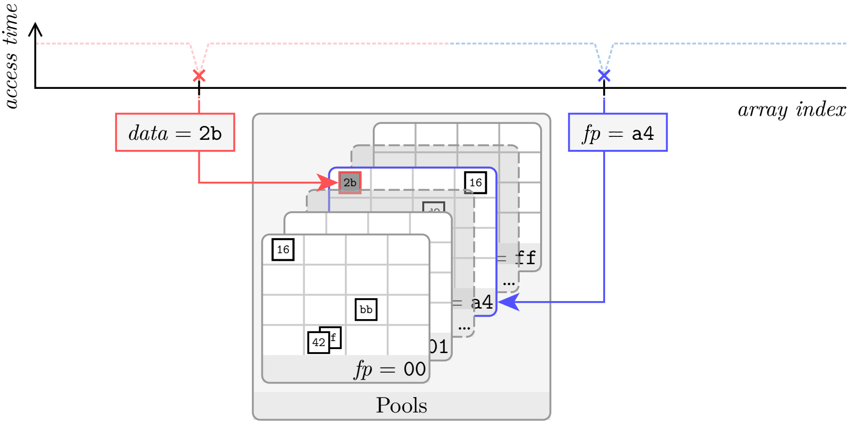
```
Key byte 15 Hyp fb Ctr:    3
Key byte 15 Hyp fd Ctr:    1
Key byte 15 Hyp fe Ctr:    1
------
[MA] Ranked Results (Top 5):
[MA] Key byte  0: 2b (  10) 5f (   9) d5 (   9) ef (   9) 0c (   8)
[MA] Key byte  1: 7e (   7) 02 (   3) 05 (   3) 72 (   3) a9 (   3)
[MA] Key byte  2: 15 (  12) d0 (   4) 11 (   3) 68 (   3) b1 (   3)
[MA] Key byte  3: 16 (  11) 06 (   3) 1a (   3) 25 (   3) 28 (   3)
[MA] Key byte  4: 28 (   7) 07 (   3) 02 (   2) 12 (   2) 17 (   2)
[MA] Key byte  5: ae (  10) 41 (   4) 60 (   3) 6e (   3) 04 (   2)
[MA] Key byte  6: d2 (   6) 28 (   2) 9a (   2) f8 (   2) 00 (   1)
[MA] Key byte  7: a6 (   8) 2b (   3) bb (   3) 18 (   2) 21 (   2)
[MA] Key byte  8: ab (  14) 3b (   3) 86 (   3) a9 (   3) 04 (   2)
[MA] Key byte  9: f7 (  14) 95 (   5) 3a (   4) 73 (   4) 00 (   3)
[MA] Key byte 10: 15 (  18) 3d (   3) 3e (   3) 45 (   3) 58 (   3)
[MA] Key byte 11: 88 (  13) 15 (   4) 79 (   4) 04 (   3) 21 (   3)
[MA] Key byte 12: 09 (  16) 5b (  13) 00 (  12) 0d (  12) 0e (  12)
[MA] Key byte 13: cf (  14) 1e (   3) 00 (   2) 14 (   2) 17 (   2)
[MA] Key byte 14: 4f (  13) 13 (   3) 36 (   3) 77 (   3) 87 (   3)
[MA] Key byte 15: 3c (   9) 6d (   4) 06 (   3) 9e (   3) fb (   3)
[MA] Collected 60000 samples in 7.659376 seconds (7833.536309 samples/s).
[MA] Process execution finished
#
[0] 1:attacker*
```

1. ZombieLoad attack (Schwarz et al., 2019)
2. Differential Attack
3. **Cache Line Fingerprinting**
4. Countermeasures
5. Conclusion

Cache Line Fingerprinting (1)

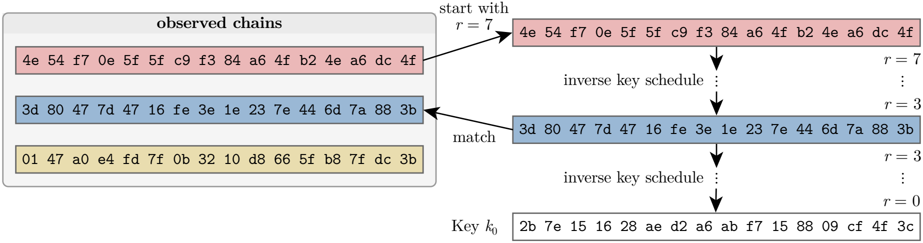


Cache Line Fingerprinting (2)



Cache Line Fingerprinting on OpenSSL

Identifying the AES key



Cache Line Fingerprinting on OpenSSL

Experimental results

CPU	No. of samples	Avg. duration (s)	Full key recoveries	
i7-2620M	100,000	12.1	9/10	(90%)
i5-4300M	100,000	11.3	3/10	(30%)
E3-1270v6	100,000	10.9	0/10	(0%)
i7-8650U	100,000	12.1	0/10	(0%)

- With variant 2, the transient execution window is too small to calculate and leak the fingerprint.

Cache Line Fingerprinting on OpenSSL

Comparison to the Domino attack

Advantages over the Domino attack:

- Samples have an additional distinctive property (fingerprint)
  - Result: 256 frequency distributions (instead of one), with less noise in each of them
- Fingerprint and data byte are always sampled from the same cache line

Demo: Cache-Line-Fingerprinting on OpenSSL

```
[MA] Type 58:  6b  (  34)  ----------  ----------  ----------  ----------
[MA] Type 59:  df  (  32)  ----------  ----------  ----------  ----------
[MA] Type 60:  ff  (  24)  ----------  ----------  ----------  ----------
[MA] Type 61:  7f  (  33)  ----------  ----------  ----------  ----------
10 AES Round Key candidates found.
Possible Candidate: 01 6a e3 9d fd 7f fe f3 10 ee 0d 9c 7e 7f dc 3b
Possible Candidate: 5b 01 e3 9d 10 7f d2 a6 c0 74 0d 9c 7e 7f 29 2f
Possible Candidate: f0 0b 15 16 10 7f d2 a6 c0 74 0d 9c 7e 7f 4f 2f
Possible Candidate: f0 01 02 fd 10 8d ba d2 c0 54 0d 9c 7e 7f 29 2f
Possible Candidate: 20 01 73 21 b5 8d ba d2 20 fc 0d 9c 7e 7f 29 2f
Possible Candidate: 3d 80 47 7d 47 16 fe 3e 1e 23 7e 44 6d 7a 88 3b
Possible Candidate: ef 44 15 41 a8 52 5b 7f b6 71 25 3b db 0b ad 3c
Possible Candidate: 10 01 e3 9d 10 7f ba d2 10 74 0d 9c 7e 7f 29 2f
Possible Candidate: 4e 54 f7 0e 5f 5f c9 f3 84 a6 4f b2 4e a6 dc 4f
The round key
        4e 54 f7 0e 5f 5f c9 f3 84 a6 4f b2 4e a6 dc 4f
is from round 7. The key
        3d 80 47 7d 47 16 fe 3e 1e 23 7e 44 6d 7a 88 3b
occurs 4 rounds before in round 3. Your AES128 key (round key 0) is:
        2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c
Possible Candidate: ea d2 73 21 b5 8d d2 d2 31 2b f5 60 7e 8d 29 2f
[MA] Collected 100000 samples in 11.304159 seconds (8846.301613 samples/s).
[MA] Process execution finished
#
[0] 0:victim- 1:attacker*
```

Countermeasures (1)

ZombieLoad countermeasures:

- Block ZombieLoad building blocks
  - e.g. SMT, TSX
- Co-Scheduling
- Clear critical buffers on context switch

Countermeasures (2)

Attack-specific countermeasure: Masking

- Differential attack: Mask intermediate results with random, secret, frequently changing masks.
  - ⇒ Generates uniform statistical distributions.
- Cache Line Fingerprinting: Requires multiple observations of the same cache line.
  - ⇒ Frequently refreshed masking (a) hides the data value and (b) prevents assignment to a constant fingerprint.

1. ZombieLoad attack (Schwarz et al., 2019)
2. Differential Attack
3. Cache Line Fingerprinting
4. Countermeasures
5. Conclusion

Conclusion

- Differential Attack: Sampled intermediate results can be used for key recovery
  - ⇒ Protecting the key is not enough
  - Especially important for algorithms without hardware support

- Cache Line Fingerprinting: efficient mechanism for extraction of constant byte sequences

Thanks for watching!

Join CARDIS Session #4 (Thursday, 2020-11-19, 16:10–17:00 CET) for further discussion.

Contact us

Till Schlüter	Kerstin Lemke-Rust
till.schlueter@cispa.de	kerstin.lemke-rust@h-brs.de